January2020: CSE322

Offline 1: HTTP File server

Task 1:

In this problem, you need to develop a web server that can handle multiple incoming HTTP GET requests and send HTTP response message back to the client, with appropriate Content-type in the response (according to the standard http protocol). The requirements of the web server are-

- Listen on a specified port, other than the standard port 80, and accept HTTP requests.
- Handle each HTTP request in a separate thread.
- Handle HTTP version 1.1 GET requests.
- Extract filename from HTTP request and return the file or an appropriate error message. If we type "localhost:port/*path*" in the address bar of a browser then:
    1. If *path* is a directory, then generate an html page showing the list of files inside as links (for differentiability purpose, you must show the directory names with **bold fonts**). Each element in the list must be a link. If we click on a directory link, server enters into that directory and shows the list of files in the new directory. If we click on a file link, that file must be downloaded.
    2. If *path* is a file, then enforce downloading it to the browser by specifying the content-type in the response message. (**Must send the file in fragments of specific chunk size**)
    3. If *path* is not found, then generate a 404: Page not found error message to the browser and also in the console of the server.
- Return a HTTP response message and close the client socket connection.
- Return appropriate status code, e.g. 200 (OK) or 404 (NOT FOUND), in the response.
- Determine and send the correct MIME type specifier in the response message.
- Your web server must be able to serve other HTTP requests while a file download is going on.
- You need to generate an appropriate log file for the corresponding http request. The log file will contain both the http request and also the http response messages.

Task 2:

Here, you will have to implement a "**Client**" program that will connect to the "**Web Server**" of Task 1. It will only be used to upload files to the "**root**" directory of the server. You have to implement it in such a way that a client can upload multiple files in parallel. So, the requirements of the client are:

- Connect to the specified port of the web server.
- Take file name as input from the console.
- Handle each file upload request in a thread.
- Upload the file to the root directory of the server in fragments of specific chunk size.
- If the given file name is invalid, write an error message to both the console of the server and the client.

Bonus Task:

Additionally, you can implement the HTTP **PUT** method besides **GET** method. The requirements of the bonus task are:

- Add a file browser to the root directory page of the web server.
- Submit the PUT request through a button.
- Upload the selected file to the root directory of the web server.
- Return a HTTP response message and close the client socket connection.

Implementation:

HTTP **GET** request message to the web server will be of the form:

“GET /... HTTP/1.1”

Make the file upload request message from the client to the server of the form:

“UPLOAD *filename*”

You can use the above information to figure out how to handle the requests from the web browser and the java client process as per the requirements.

Warnings:

- You need to implement the basics of http protocol accordingly; therefore, **you cannot use any JAVA high level library. You cannot use Javascript. All the operations should be strictly limited to Socket Programming and html manipulation.**
- Remember that for each socket the input and output stream can be instantiated only once.
- Please do not copy. If found guilty, you will be given a straight negative 100% marks.

Help: https://www.tutorialspoint.com/http/http_methods.htm

Tentative Mark Distribution:

Task 1	Show Directory	5
	File download	6
	Error: Not found	2
	Parallel requests	3
	Log file	2
Task 2	File upload	5
	Error message	2
	Parallel upload	3
Bonus	File upload	5
Task 0	Proper submission	2
	Total	30+5

Submission Instruction & Deadline:

- Put your source codes in a folder named “1605xxx”, then zip it into “1605xxx.zip” and upload to the moodle. (Your submission **must not contain** any other file)
- Deadline: **7:55 am, Sunday, March 15, 2020**.